NHTI - Concord's Community College

Division of Continuing Education

31 College Drive
Concord, New Hampshire 03301-7412

CP 107 Introduction to Programming with C++

Department: EET/CPET/BNCT

Hours per class: Lecture 2.0. Laboratory 2.0
Credits: 3
Course co requisite: None, used to be IS 101 and MT 133

Instructor: Professor Mohammad "Saleem" Yusuf
Phone: Home (603) 424-8147 Work (978)-370-8131
Email: MSaleemYusuf@Yahoo.com

Class Dates and Time
Wednesday nights, from 1/22/2008 to 5/6/2008 6:00 PM to 10:10 PM

Introduction

Required Textbook

Absolute C++ by Savitch, 2nd edition. Addison Wesley,

ISBN for the package 0-321-38166-1

It also contains a student version of Microsoft Visual C++ Compiler. You can also download MS VC++ .Net
http://msdn.microsoft.com/vstudio/express/

In this course you will learn

- How to write computer programs using C++
- Understand concepts of Object Oriented programming
- Able to write programs using structured programming and object oriented programming methods

What is a Computer?

Computer

- A device capable of performing computations and making logical decisions

Computer programs

- Sets of instructions that a computer follows to process data and make decision

Hardware

- Parts that make up a computer
  - Examples: keyboard, screen, mouse, disks, memory, CD-ROM, and processing units

Software

- Sequence of instructions in the form of computer files that controls the behavior of a computer

Programming Languages

Language that computer understands

- Used by programmers to communicate with the computer
- Which is then converted into 0's and 1's
- Computer only understands 0's and 1's

Three types of programming languages

- **Machine languages**
  - A symbolic representation of actual machine code
  - Strings of numbers that are converted into 0’s and 1’s, that provides machine, specific instructions
  - Example:

```
+1300042774
+1400593419
+1200274027
```

Machine Languages, Assembly Languages, and High-level Languages

- **Assembly languages**
  - Short, English-like abbreviations representing elementary computer operations (translated via assemblers)
  - Example:

```
LDA      BASEPAY
SUM      OVERPAY
STORE GROSSPAY
```

- **High-level languages**
  - Similar to everyday English
  - Translated via compilers into binary code
  - Language almost Machine Independent
  - Compiler is Machine (Hardware) Dependent
  - Example:

```
grossPay = basePay + overTimePay
```

High-level Languages

High-level languages

FORTRAN Formula Translation

- Used in scientific and engineering applications
- John Backus of IBM in 1954, released in 1957-58 and is still in use

COBOL Common Business Oriented Language

- Used to manipulate large amounts of data

BASIC

- Very Simple Scientific language by Microsoft. Now evolved into the most used programming language

Pascal Structured Scientific Language

- Used to teach structured programming

ADA DODs attempt to standardize software development

- Was a requirement for government contracts

High-level Languages

Prolog PROgramming LOGic – 1970s Edinburgh University

- Very popular with Artificial Intelligence (AI)

Other high level languages

- Smalltalk, Algol, LISP, Modula, Snobol

C Programming language

- Mid to high level language, replacing Pascal and Fortran
- Mostly used for system software development

Java, C++

Fourth Generation Languages

- Mostly for client server application development

  Dbase IV, FoxPro

Scripts – VBScript, JavaScript, Perl, Python

History of C and C++

C was developed at Bell Labs

- C evolved from two other programming languages, BCPL and B

ANSI C

- Established worldwide standards for C programming

C++ evolved from C

- Provides capabilities for object-oriented programming
- User defined data types
- Huge improvement over C, without losing functionality

C++ Standard Library

C++ programs

- Built from pieces called classes and functions

C++ standard library

- Rich collections of classes and functions that come with the compiler

C++ Third Party library

- Rich collections of classes and functions that you purchase from software vendors

Structured Programming

Structured programming

- Disciplined approach to writing programs
- Reusable program modules
  - Subroutine
  - Function
  - Class
- Clear, easy to understand, easy to test and debug, and easy to maintain

Developing a C++ Application

Steps in developing a C++ Application

Write Code → Compile Code → Link Code → Execute Code

Write Code

- Program is created in an editor and stored on the disk
- Output is a text file (cpp, .h)

Compiler

- A tool that checks the syntax of the code and converts computer program into a binary file. This code is known as object code.
- Compiler makes two passes to compile a program. First pass is called pre-processor pass. Variables and methods whose address cannot be resolved on the first pass are resolved in the second pass
- Output of a compile operation is an object file (.obj)
- If there are syntax errors, user must go back and fix the code first and then re-compile the program

Developing a C++ Application

Linker

- A tool that combines the object code with built-in library's object code (that you include in your program). It creates a map of all the code and assigns addresses to different parts of code like functions and modules.
- Output is an executable file (.exe)

Build

- To compile and link the program and create .exe file

Run

- To execute a program

Clean

- Delete all intermediate files. Forces the compiler to recompile all the code. If compiler finds that code has not changed, it does not recompile code. Clean forces it to recompile

A Sample C++ Program (1 of 2)

Display 1.1 A Sample C++ Program

```
1  #include <iostream>
2  using namespace std;

3  int main( )
4  {
5      int numberOfLanguages;

6      cout << "Hello reader.\n"
7           << "Welcome to C++.\n";

8      cout << "How many programming languages have you used? ";
9      cin >> numberOfLanguages;

10     if (numberOfLanguages < 1)
11         cout << "Read the preface. You may prefer\n"
12              << "a more elementary book by the same author.\n";
13     else
14         cout << "Enjoy the book.\n";

15     return 0;
16 }
```

Basic Parts of a Program

Libraries and Namespaces

- Include libraries supplied by the compiler vender
- Name spaces allows you to include code within a named region (instead of including all). Also allows you to create your own namespace

Start and End

- All console programs start with function **main**
- All blocks of code start with **{**
- All blocks of code end with **}**
- There should always be matching **{** and **}**
- All functions start with a function name and then **{**
- All functions (should) end with return and **}**

Basic Parts of a Program

Statements

- A line of code
- Must end with a semicolon character
- Can span across multiple lines

Examples of Statements

- Declare Variables and optionally assign value to it
- Expressions, assignment, calls to a function
- Console Input (Ex: read input from user and pass info to program)
  - Read text from the windows console
  - cin or std::cin followed by >> (stream extraction operator)
- Console Output (Ex: take info from program and display it on monitor)
  - Write text to the windows console
  - cout or std::cout followed by << (stream insertion operator)

Basic Parts of a Program

Comments

- Compiler ignores comments
- Only meant for users to improve program readability.
- Two types
  - **/*** some text ***/** all code between /* and */ is comment
- **//** all code after // is comment . The text becomes comment until end of line.
- **//** can be in any column.
- **/*** text ***/** all code between /* and */ is comment. */ means the end of comment. **/*** and ***/** do not have to be on the same line
- /* */ is an old way to comment code. It is prone to errors and is not the preferred way to comment your code

```
// This is a comment.

/* Old way of writing a
comment. Can span multiple
lines. Everything is a
comment until
*/

#include <iostream>
using std::cout;
using std::endl;

int main()
{
    cout << "Hello World";
    cout << endl;

    return  0;
}
```

Comments

Text between /* and */ or text following a //.

Compiler ignores comments. It is to improve program readability.

preprocessor directive

An instruction to the C++ to the compiler to process this on the first pass (preprocessor).

Lines beginning with # are preprocessor directives. # does not have to be in row 1

#include <iostream> tells the compiler to include the contents of the file <iostream>, which contains I/O routines for reading and writing to the console screen.

C++ programs contain one or more functions, one of which must be **main**

Parenthesis are used to indicate a function

int means that **main** "returns" an integer value. More in Chapter 3.

A left brace { begins the body of every function and a right brace } ends it.

Printing a Line of Text

```
std::cout << “Welcome to C++!\n”;
```

`std::cout`

- Standard output stream object
- “Connected” to the screen
- **`std::`** specifies the "namespace" which **`cout`** belongs to
  - **`std::`** can be removed through the use of **`using`** statements

`<<`

- Stream insertion operator
- Value to the right of the operator (right operand) inserted into output stream (which is connected to the screen)

`\`

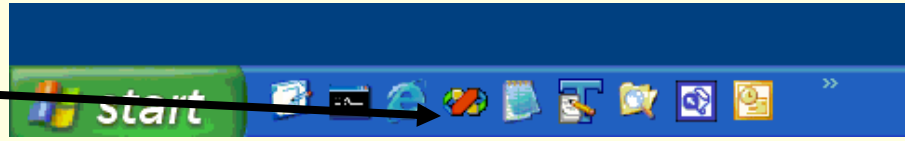
- Escape sequence character
- Indicates that a “special” character is to be output

Escape Sequences

Escape Sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
\a	Alert. Sound the system bell.
\\	Backslash. Used to print a backslash character.
\"	Double quote. Used to print a double quote character.

Using Dev Studio

Start MS VC++ 6.0 from the Taskbar bar by clicking on the icon

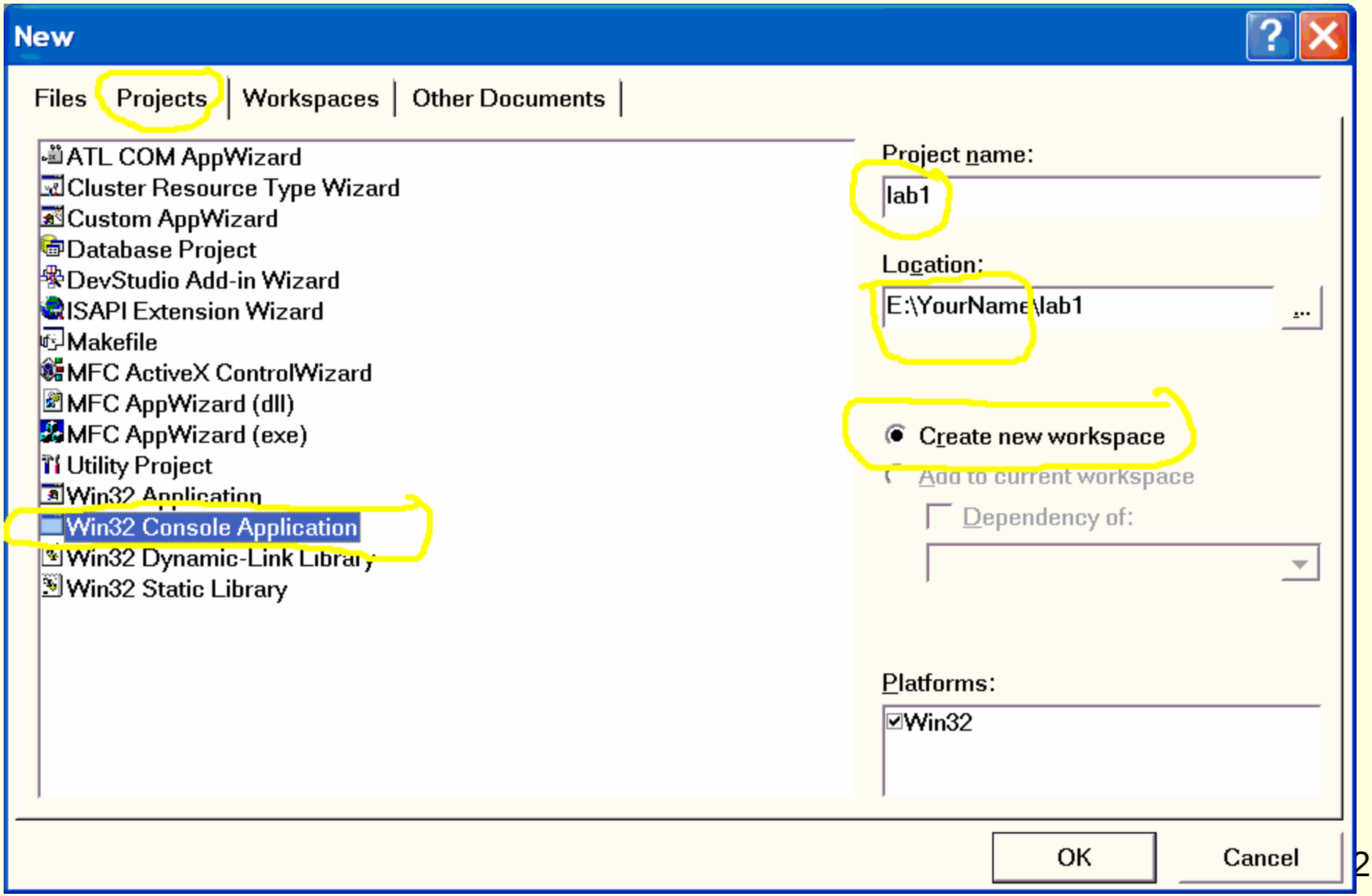


Or Start MS VC++ 6.0 by clicking on the Start button, then selecting "All programs", and then selecting Microsoft Visual C++ 6.0 and then again selecting "Microsoft Visual C++ 6.0"

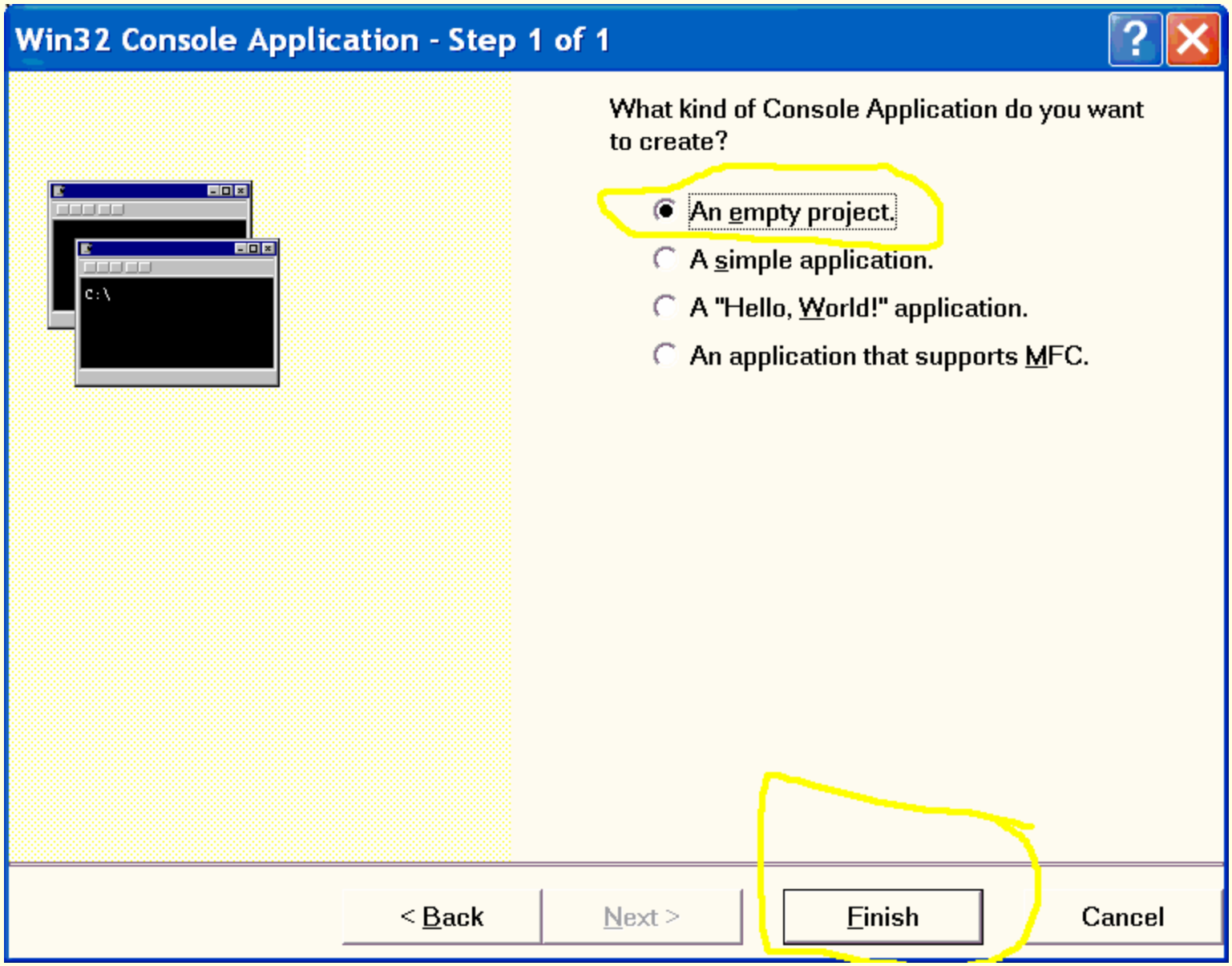
Creating a New Project

- Create New Project
  - From the "**File**" menu, select "**New**". This will bring up the "New" dialog box.
  - In the "New" dialog box, click on the "**Projects**" tab and make selections that are highlighted in the picture in the slide "New Project" .
  - Then click on the OK button.
  - A Dialogbox will appear, asking your "Which kind of Cosole Application you want to create?" .
  - Select "an empty project". This is the only kind of project we will create in this class. Click on "**Finish**" button.
  - Another Dialogbox will appear that will provides information about your project. Click **OK** to complete your project.
  - You are now back to MS VC++ 6.0. The next step is to add a C++ source file to your project.

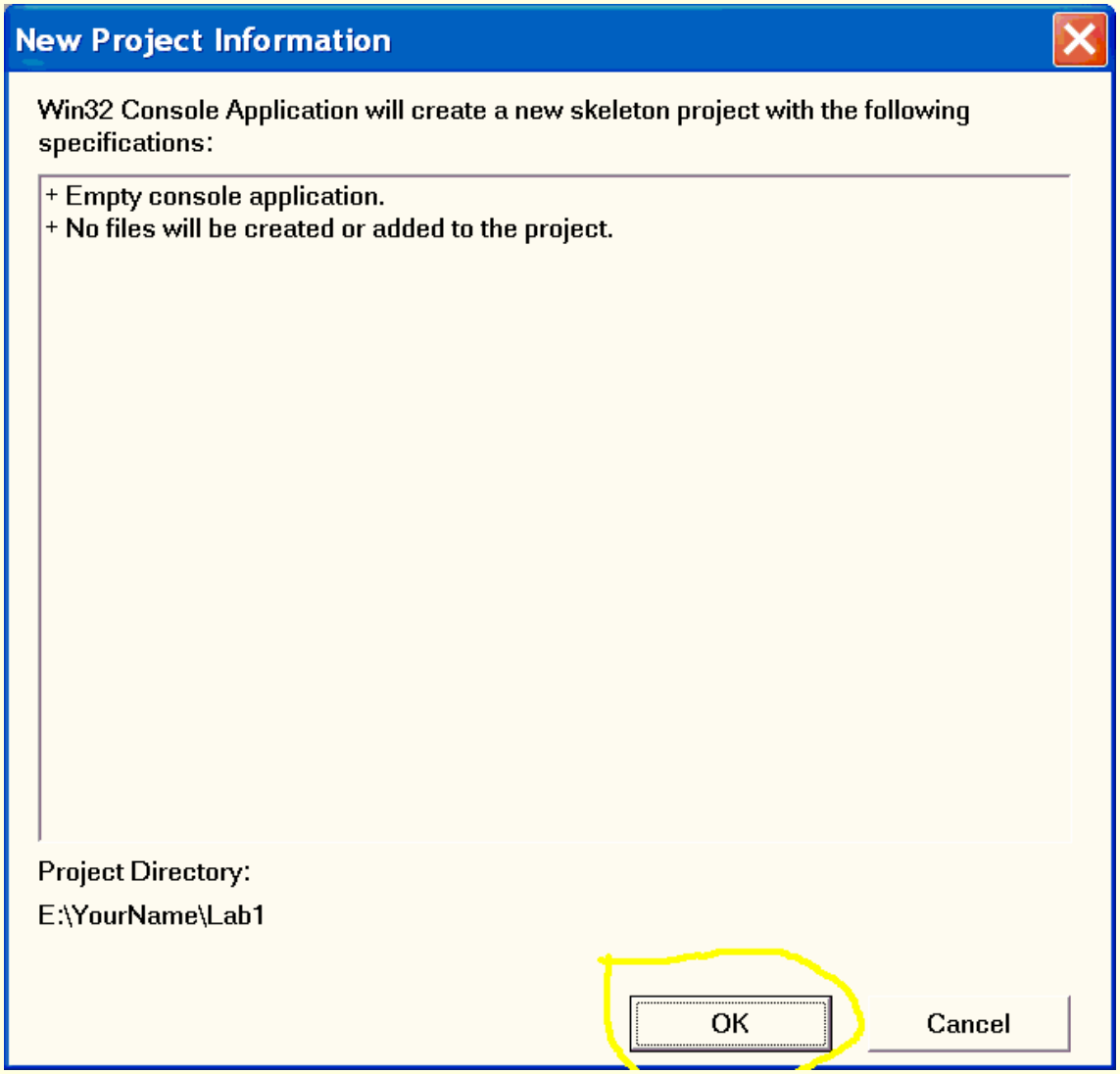
New Project


New
Files
Projects
Workspaces
Other Documents
ATL COM AppWizard
Cluster Resource Type Wizard
Custom AppWizard
Database Project
DevStudio Add-in Wizard
ISAPI Extension Wizard
Makefile
MFC ActiveX ControlWizard
MFC AppWizard (dll)
MFC AppWizard (exe)
Utility Project
Win32 Application
Win32 Console Application
Win32 Dynamic-Link Library
Win32 Static Library
Project name:
lab1
Location:
E:\YourName\lab1
...
Create new workspace
Add to current workspace
Dependency of:
Platforms:
Win32
OK
Cancel

Empty Project



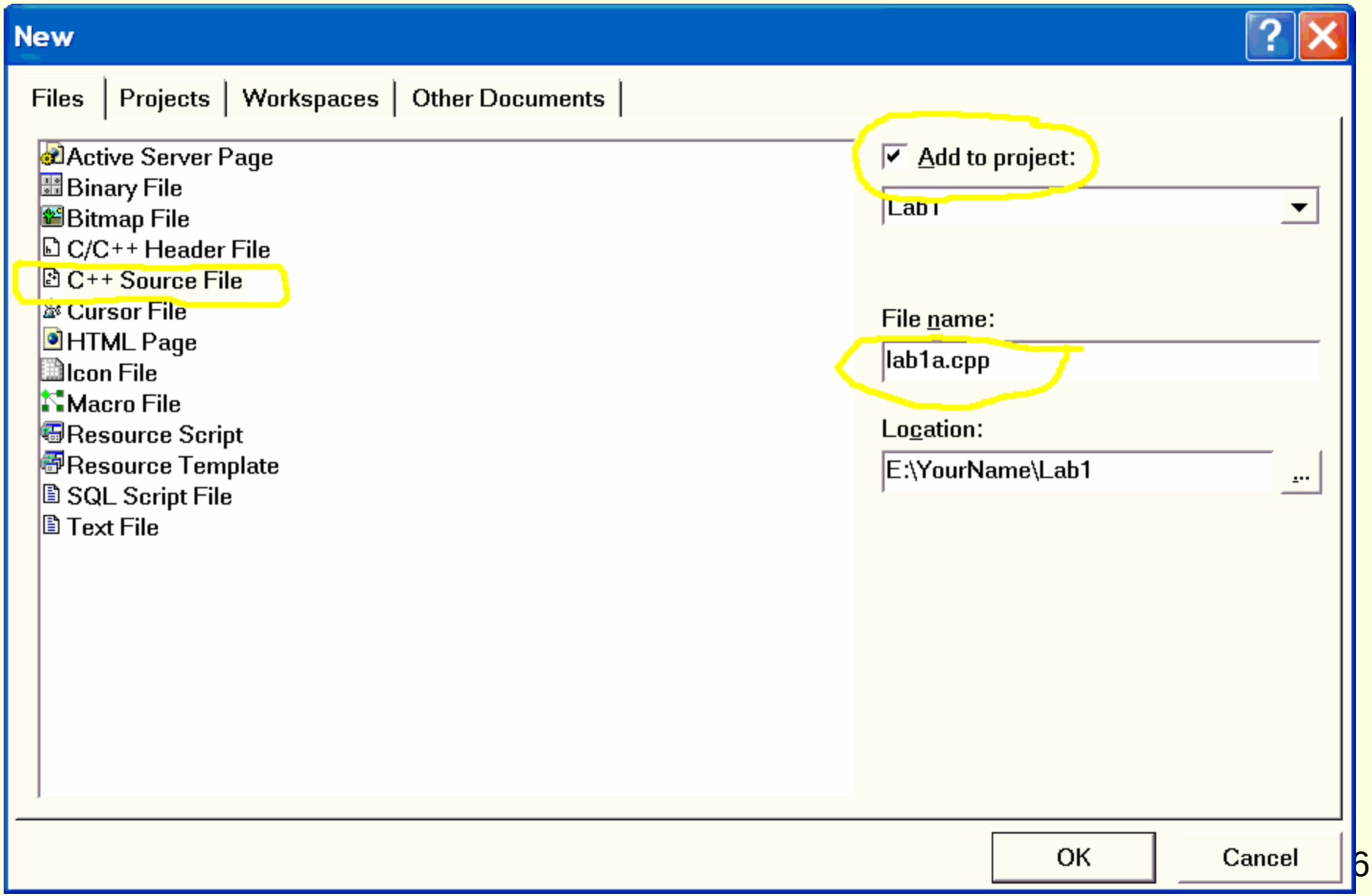
Confirm Empty Project



Adding a File to the Project

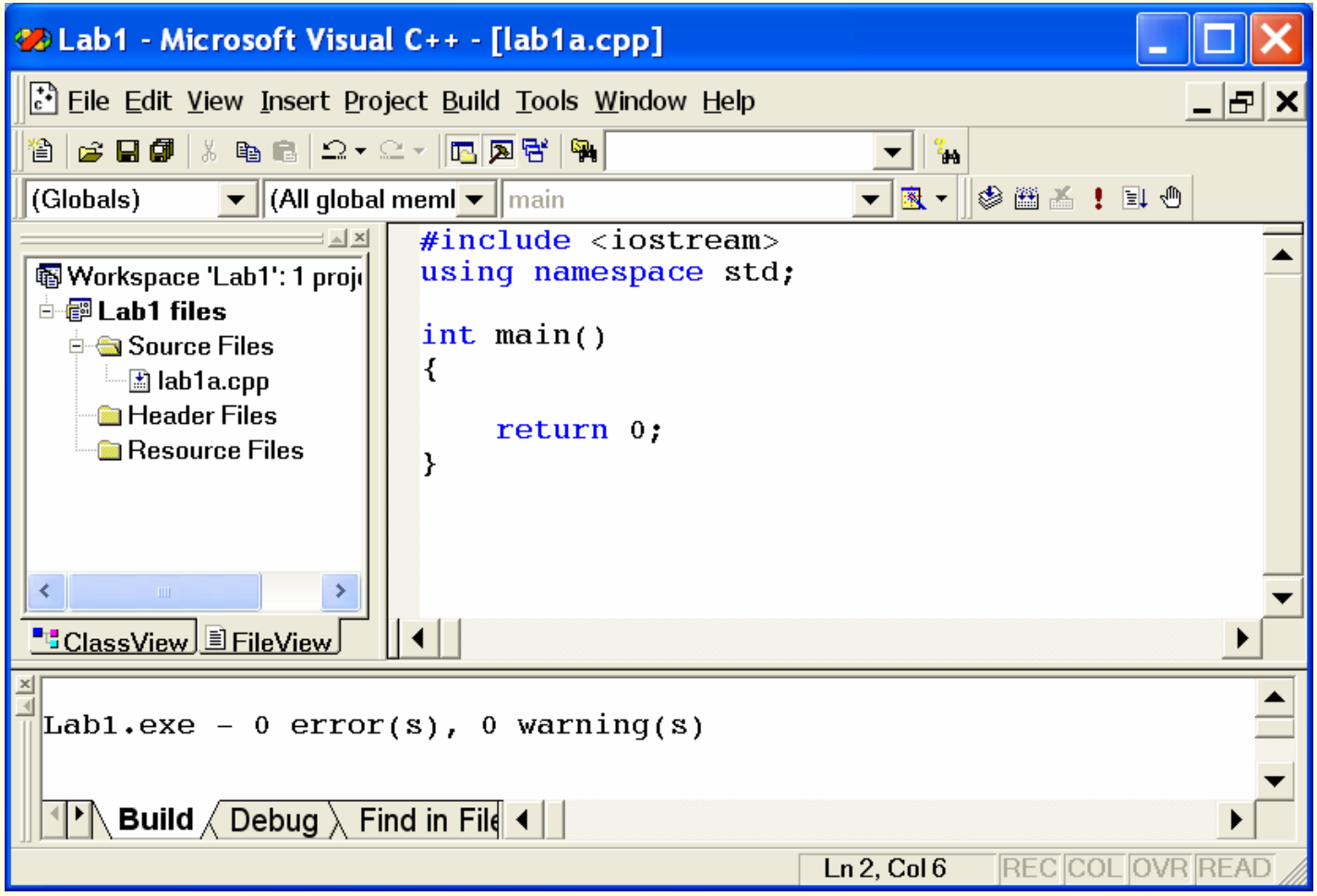
- Add a New Source File to the Project
  - From the “**File**” menu, select “**New**”. This will bring up the “New” dialog box.
  - In the “New” dialog box, click on the “**Files**” tab and make selections that are highlighted in the picture in the slide “Adding a New File” .
  - Then click on the OK button.

Adding a New File

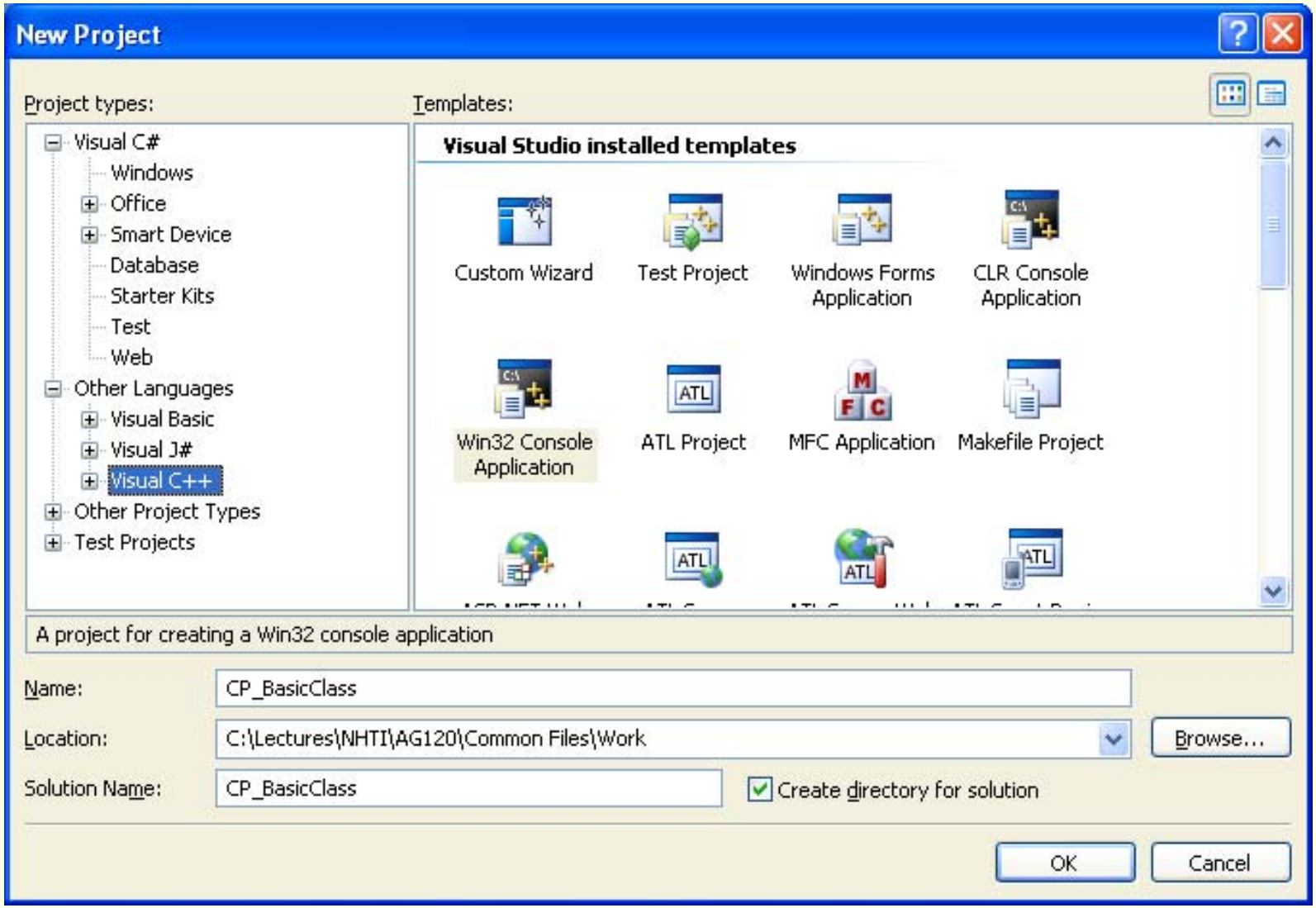


A template for All Programs

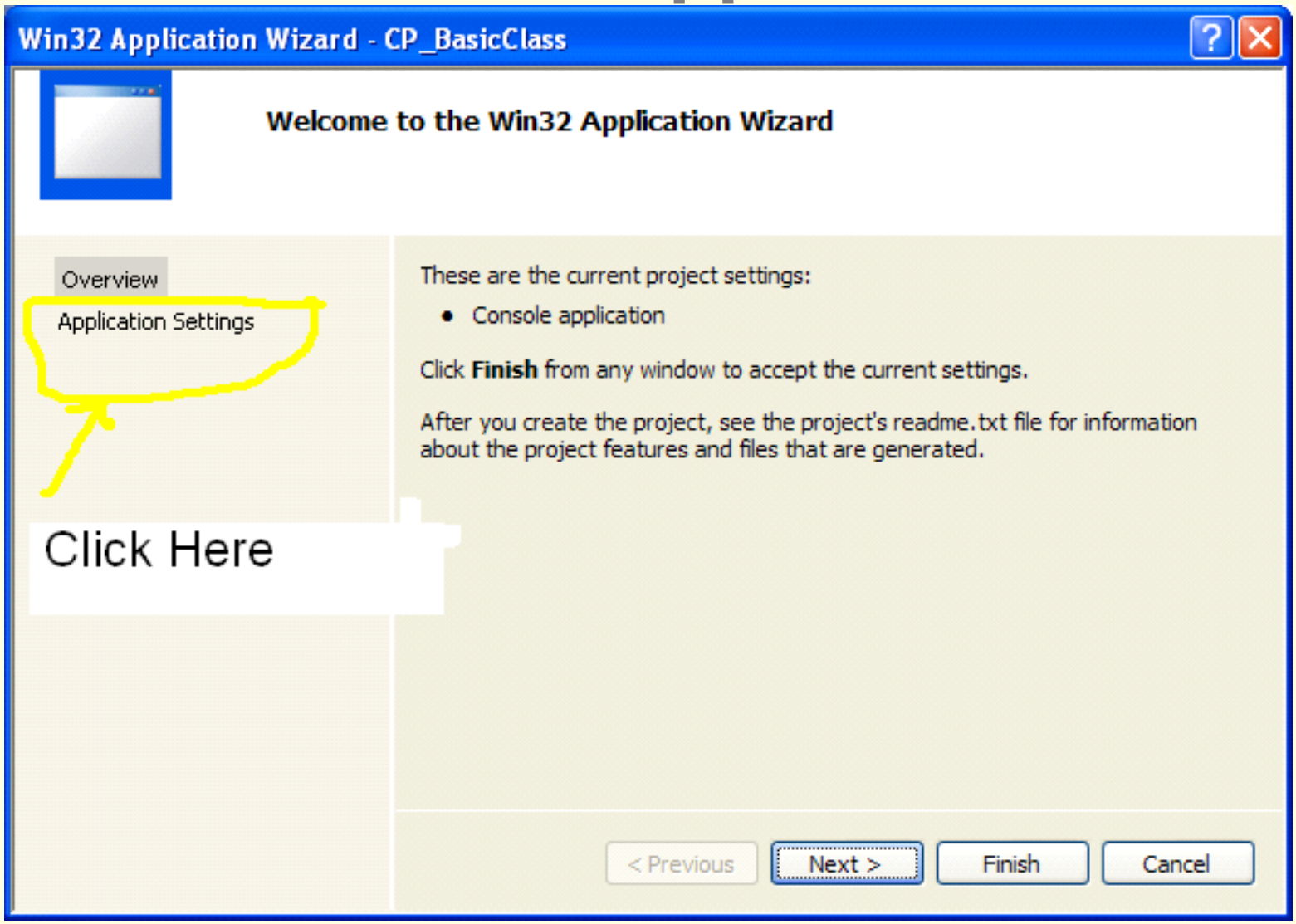
After you add the file, type in minimal code as displayed. You will need this for almost all assignments. This is what Dev Studio should look like.



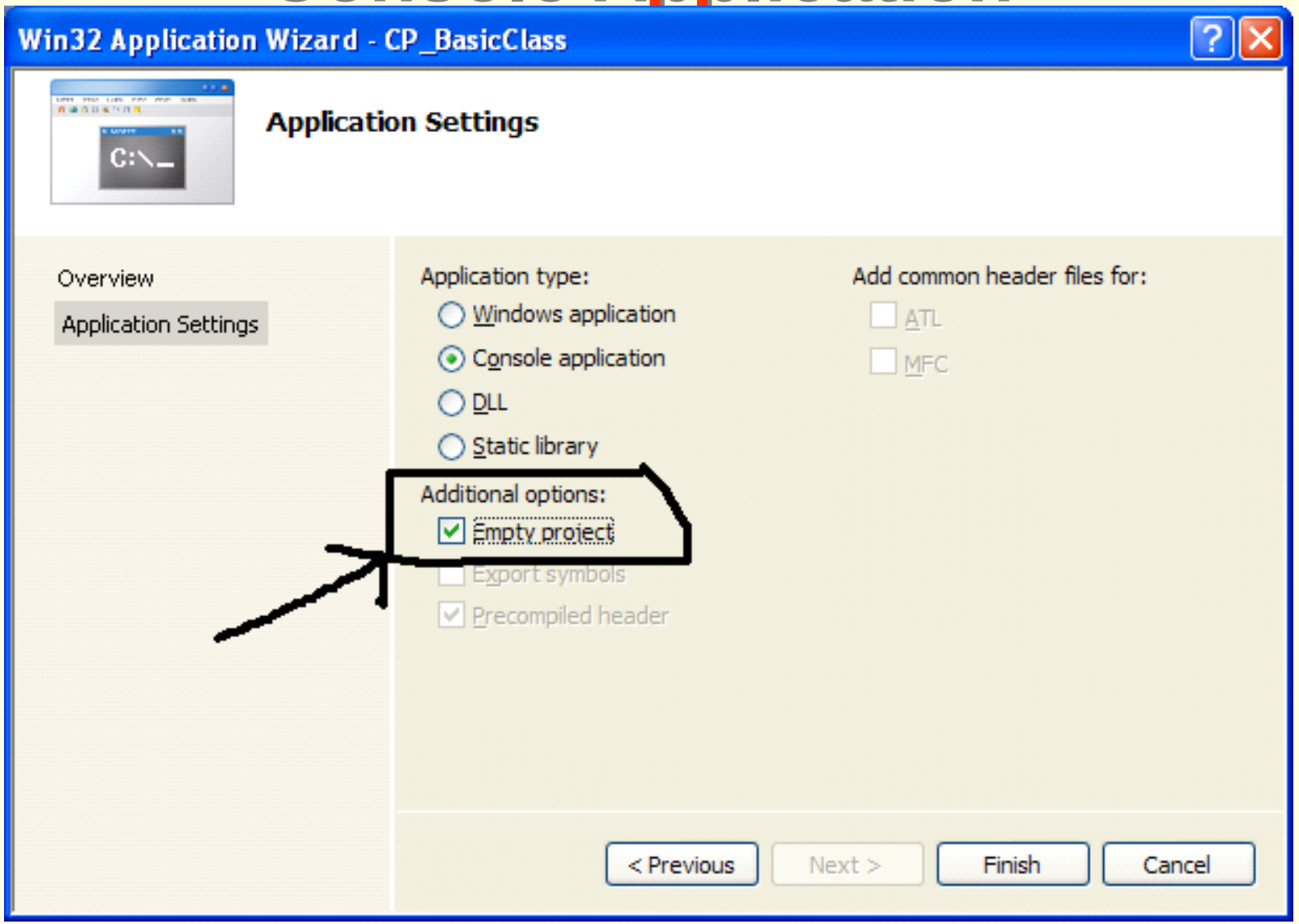
Getting Started – A Simple Console Application



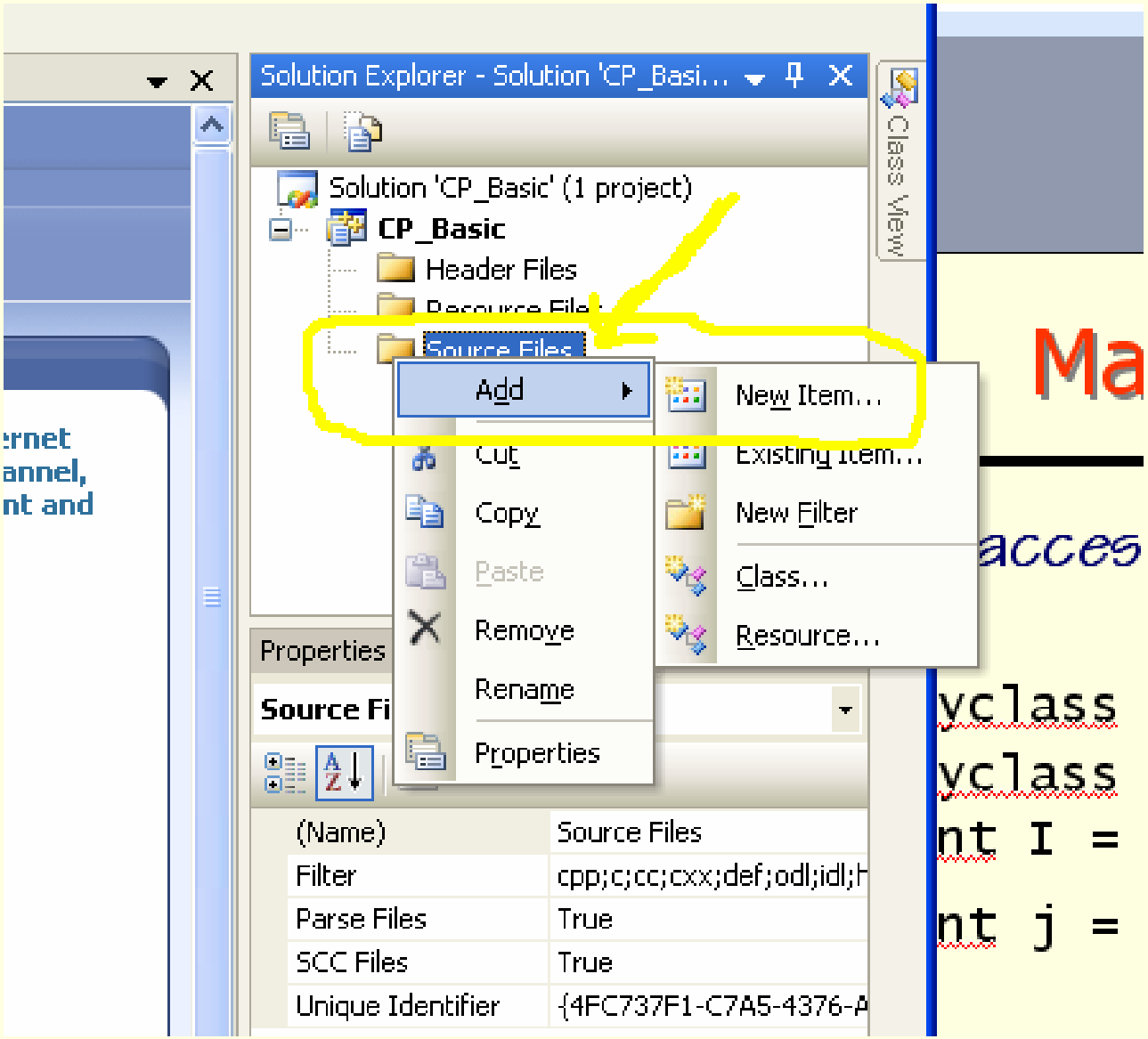
Getting Started – A Simple Console Application



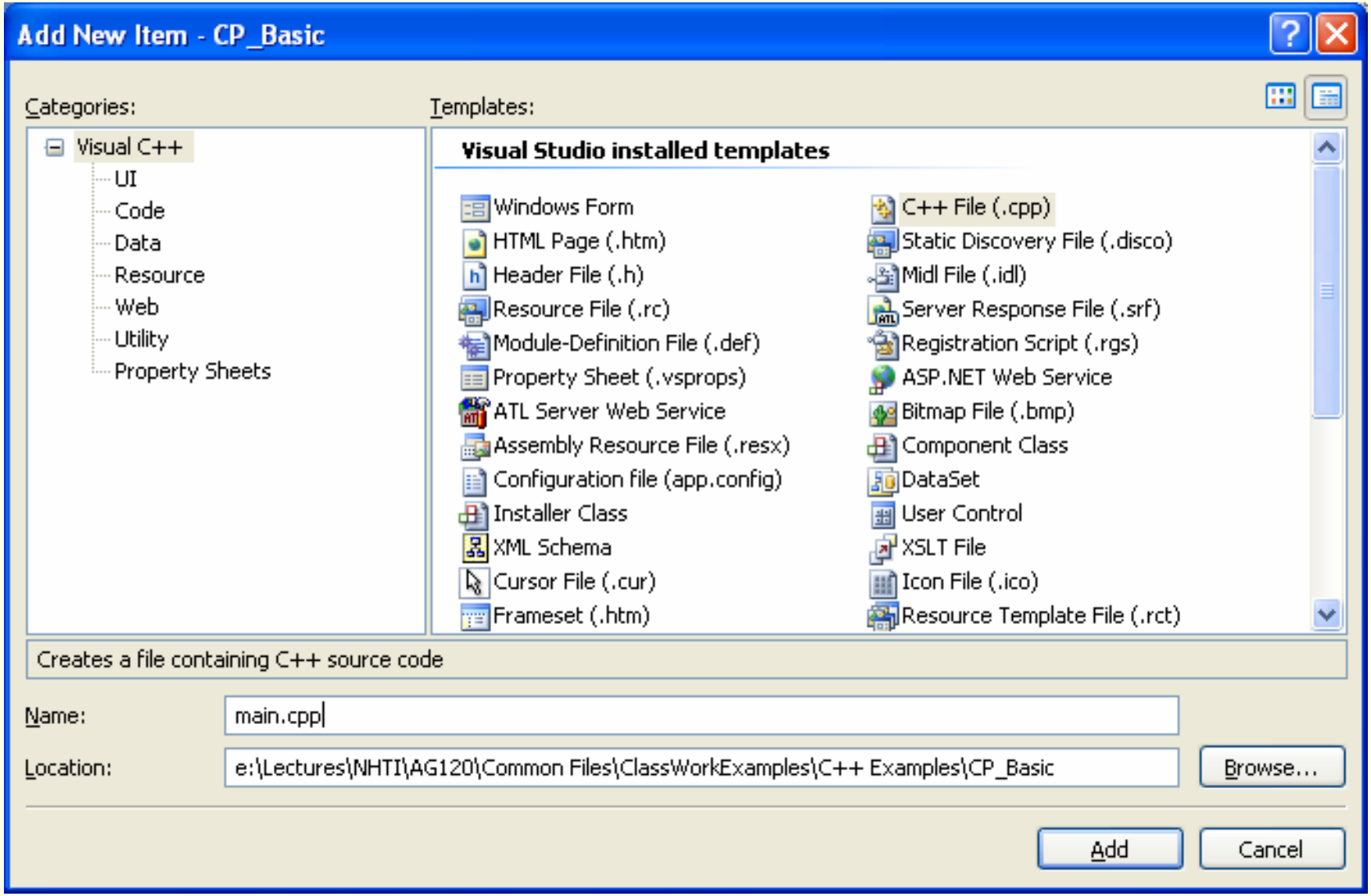
Getting Started – A Simple Console Application



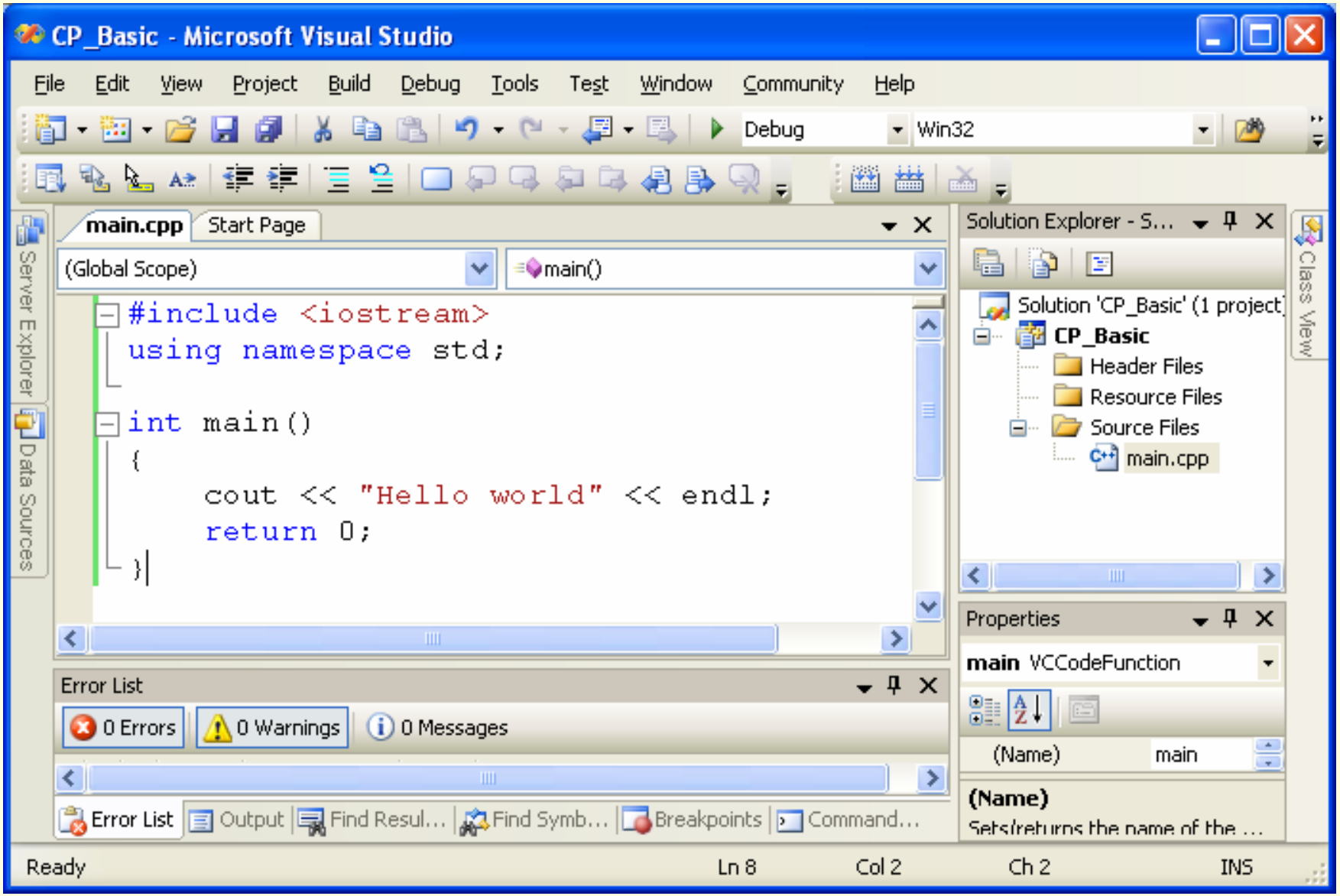
Add Source File



Select C++ File



Write Simple Hello Word Program



Using statement, Namespace

`using` statements

- Eliminate the need to use the **`std::`** prefix
- Allow us to write cout instead of **`std::cout`**
- To use the following functions without the **`std::`** prefix, write the following at the top of the program

```
using std::cout;
using std::cin;
using std::endl;
```

- Instead of including individual functions, you can include the whole namespace.

```
Using namespace std;
```

Memory Concepts

C ++ Identifiers

- User defined or Compiler defined names
- Name you give to your variables, functions etc.
- Make sure that you do not use keywords or names reserved by the compiler

Variables

- A name given to a location of memory where data can be stored for use by a program
- Every variable has a name (you give), a type, a size and a value
- Whenever a new value is placed into a variable, it replaces the previous value - it is destroyed
- Reading variables from memory does not change their data

C++ Data Types &Variables

Variable

- A memory location to store data
- Must be declared with a **name** and a data type before they can be used
- variable may be defined anywhere, as long as it is defined before it is used
- Identifiers must begin with a character or underscore which can then be followed by characters, underscores, and numbers

C++ Data Types & Variables

- A data type may be either machine (system) defined or user defined
- System defined data type is also known as primitive data type
- Machine defined type such as **`integer`** allow you to characterize both the kind of data as well as permissible operations that can be performed on the data
  - Integers have a size of 32 bit
    - Only specific operations can be performed on Integers (+,-,=,==,…)
    - Integers can be created and destroyed

Primitive Data Types

C++ supports 7 Primitive (Basic) Data Types

Data Type	Memory used	Description
int	32 bits/4 bytes	Integer
float	32 bits/4 bytes	Single Precision floating point number
double	64 bits/8 bytes	Double Precision floating point number
char	8 bits/1 byte	Character
void	N/A	“nothing” used to state that a function returns no value
wchar_t	16 bits/2 bytes	wide character, used to represent characters in international character sets
bool	8 bits/1 bytes	boolean (used to store “true” or “false”)

```
#include <iostream>
   using namespace std;
   int main()
   {
      int  i = 10;         // declare an int & initialize it
      double x = 10.556; // declare a double & initialize it
      float y = 20.379F; // declare a float & initialize it
      char ch = ‘a’;      // declare a char and initialize it
      bool done = true;  // declare a char and initialize it

      // Display the value variable
      cout << "Value of i, x and y  is " << i
                << “, “ << x << “, “ << y << endl;

      // Display the value variable
      cout << "Value of ch, and done is "
                << ch << “, “ << done << endl;
      return  0;
   }
```

Integers and Characters

- int is used to designate a signed, whole number.
  - Through the use of a modifier, they can be used to represent unsigned numbers

char and wchar_t are used to hold characters (char) or long characters (wchar_t)

- Internally, all chars are treated as integers. Each character is represented by a unique integer.
- It is possible to perform math operations on chars, but be careful about using them in loops because of their limited size

Floats and Doubles

float and double are used to hold signed numbers having a whole part and a fractional part (e.g. 3.5)

- floats and doubles can represent data in the form of an exponent and mantissa
- Exponent is used to maintain the location of the decimal point
- Mantissa contains the number with the decimal point "normalized". This means that there is a known method to construct the number using the exponent
- For example, the number 1.5 can be represented as .15 E+1 or 15.0E-1 (e can be upper or lower)
- When floats or doubles are used in arithmetic, the machine processes the exponents and mantissas to produce a resulting number

Modifiers

Modifiers can be applied to the predefined data

Type	Memory used	Range
char (default)	8 bits	-128 to 127
unsigned char	8 bits	0 to 255
int (default)	16 bits	-32,768 to 32,767
unsigned int	32 bits	0 to 65,535
short int	16 bits	-32,768 to 32,767
long int	32 bits	-2,148,483,648 to 2,147,483,647
unsigned long int	32 bits	0 to 4,294,967,295
double	64 bits	1.7E-308 to 1.7E+308
long double	64 bits	1.7E-308 to 1.7E+308

Prefix and Suffix for constants

U = unsigned

L = long

F = float

```
char chr = 'a';
wchar_t wc = L'a';

int myInt = 300;          //OK
int myInt = 300UL;        //OK, unsigned long

float myFloat = 40.500;  // NOT OK, default is double
float myFloat = 40.500F;  // OK
```

C++ Operators

C++ Operators

- Arithmetic
- Relational
- Logical
- Bitwise and other operators

```
+      -      *      /      %      ^      &
|      ~      !      =      <      >      +=
*=     /=     %=     ^=     &=     |=     <<
>>     >>=    <<=    ==     !=     <=     &&
||     ++     --     ->*    []     ()     ->
new    delete ,
```

C++ Operators

Arithmetic

+ Plus, - Minus, / Divide, * Multiply, % Modulus

Relational

- Tests the relationship between two data types, returns true/false

  > Greater than, >= Greater or Equal to,

  < Lower than, <= Lower than or Equal to

  == Equal to, != Not equal to

Logical

- Tests the logical relationship between two operands

  && AND, || OR, ! NOT

Relational and Logical operators always return a boolean

(0 == false, non-zero == true)

C++ Arithmetic Operator

Arithmetic Operators

```
+ Plus,      - Minus,    * Multiply,
% Modulus, / Divide,
++ increment, -- decrement
```

- Binary operations

```
result = operand1 operator operand2
a = b + c;
x = 5 * 4;
```

- Unary operations

```
x++;  //increment operator, postfix
--x; //decrement operator, prefix
```

```
#include <iostream>
using namespace std;
int main()
{
     int  i, j, iSum;       // declare vars, integers
     i = 10;                // assign a value to an int
     j = 20;
     iSum = i + j;          // put sum of i and j in iSum
     // display the values
     cout << i << "+" << j << " = " << iSum << endl;

     double x, y, dSum;   // declare variables, doubles
     x = 3.5;             // assign values
     y = 20;
     dSum = x * y;        // put product of x and y in dSum
     // display the values
     cout << x << "*" << y << " = " << dSum << endl;
    return  0;
}
```

C++ Arithmetic Operators

Arithmetic calculations

- Use `*` for multiplication and `/` for division
- Integer division truncates remainder
  `7 / 5 evaluates to 1`
- Modulus operator returns the remainder
  `7 % 5 evaluates to 2`

Operator precedence

- Some arithmetic operators act before others (i.e., multiplication before addition)
  - Be sure to use parenthesis when needed
- Example: Find the average of three variables a, b and c
  - Do not use: **`a + b + c / 3`**
  - Use: **`(a + b + c ) / 3`**

C++ Arithmetic Operators and precedence

Arithmetic operators

C++ operation	Arithmetic operator	Algebraic expression	C++ expression
Addition	+	f + 7	f + 7
Subtraction	-	p – c	p - c
Multiplication	*	bm	b * m
Division	/	x / y	x / y
Modulus	%	r mod s	r % s

Rules of operator precedence

Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are everal pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.
, /, or %	Multiplication Division Modulus	Evaluated second. If there are several, they re evaluated left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Precedence

Highest	++, --
…….	- (unary minus)
…….	*, /, %
Lowest	+, -

- Operators of same precedence are evaluated left to right
- Both relational and logical operators are lower in precedence than arithmetic operators.
- Efficient to use shorthand --, ++ (then x = x - 1)

```
x = x + 10 can be written as x += 10
```

Precedence

- The precedence of operators in C++ is the same as C.
- Never get in the habit of relying on precedence !! Use parentheses !!

Remember, Arithmetic operators have higher precedence

- (x > y + 2) is equal to (x > (y + 2))
- 10 > 1 + 2 is 10 > (1 + 2) which returns a true

Stream Insertion and Extraction operators

>> (stream extraction operator)

- When used with **`std::cin`**, waits for the user to input a value and stores the value in the variable to the right of the operator
- The user types a value, then presses the *Enter* (Return) key to send the data to the computer
- Example:

```
int myVariable;
std::cin >> myVariable;
```

  - Waits for user input, then stores input in **`myVariable`**

= (assignment operator)

- Assigns value to a variable
- Binary operator (has two operands)
- Example:

```
sum = variable1 + variable2;
```

```
#include <iostream>
using namespace std;
int main()
   {
     int  i, j, iSum;        // declare an ints
     double x, y, dSum;     // declare a doubles
     //Prompt user for input
     cout << “Enter two integer numbers “ << endl;
     cin >> i, j;       // put first value in i and second val in j
     iSum = i + j;      // put sum of I and j in iSum
     cout << i << “+” << j << “ = “ << iSum << endl;
     //Prompt user for input
     cout << “Enter two floating point numbers “ << endl;
     cin >> x, y;       // put first value in x and second val in y
     dSum = x * y;      // put sum of I and j in iSum
     cout << x << “*” << y << “ = “ << dSum << endl;
   return  0;
   }
```

Equality and Relational Operators

`if` structure

- Test conditions truth or falsity. If condition met execute, otherwise ignore

Equality and relational operators

- Lower precedence than arithmetic operators

Table of relational operators on next slide

Selection Structures – if / else

```
int grade = 90;
if ( grade >= 90 )
{
   cout << “You have got an A, “;
}
else if ( grade >= 80 )
{
   cout << “You have got a B, “;
}
else
{
   cout << “You have passed, “;
}
cout << “congratulations!!“ << endl;
// You have got an A, congratulations!!
```

Selection Structures – if / else

```
int grade = 80;
if ( grade >= 90 )
{
   cout << “You have got an A, “;
}
else if ( grade >= 80 )
{
   cout << “You have got a B, “;
}
else
{
   cout << “You have passed, “;
}
cout << “congratulations!!“ << endl;

// You have got a B, congratulations!!
```

Equality and Relational Operators

Standard algebraic equality operator or relational operator	*C++ equality or relational operator*	*Example of C++ condition*	*Meaning of C++ condition*
Relational operators			
>	**>**	**x > y**	**x** is greater than **y**
<	**<**	**x < y**	**x** is less than **y**
≥	**>=**	**x >= y**	**x** is greater than or equal to **y**
≤	**<=**	**x <= y**	**x** is less than or equal to **y**
Equality operators			
=	**==**	**x == y**	**x is equal to y**
≠	**!=**	**x != y**	**x is not equal to y**

Arithmetic Operators

Arithmetic operators

C++ operation	Arithmetic operator	C++ expression
Addition	+	f + 7
Subtraction	-	p - c
Multiplication	*	b * m
Division	/	x / y
Modulus	%	r % s

Rules of operator precedence

Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.
*, /, or %	Multiplication Division Modulus	Evaluated second. If there are several, they re evaluated left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Sited sources & websites

http://www.spellen.org/youcandoit/glossary.htm

Slides created from slides or information obtained from
Prentice Hall Prentice
And
Deitel & Deitel and Pearson Addison-Wesley.

End

End of lecture 1

Click on the following link to return to the website

http://www.geocities.com/MSaleemYusuf/lecture_1.htm

http://mysite.verizon.net/MSaleemYusuf/lecture_1.htm